VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Java herní portál

Java game portal

2017 Ondřej Kolář

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Ondřej Kolář**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Java herní portál
Java Game Portal

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem je vytvořit internetový herní portál umožňující hraní již vytvořených her v jazyce Java. Součástí portálu je webové rozhraní na platformě JavaEE a také rozhraní pro platformu JavaSE. Každé z těchto rozhraní bude nabízet jinou funkcionalitu.
Systém umožní:
1. Správu uživatelů.
2. Správu dostupných her.
3. Správu odehraných her.
4. Vytváření turnajů (každý s každým, pavouk).
5. Rovněž bude navrženo a použito rozhraní na již existující hry, které bude umožňovat:
   a) zakládat nové hry,
   b) získávat průběžné obrázky probíhající hry, bodový stav, pořadí, atd.
6. Připojování hráčů a jejich znovu připojení po ztrátě spojení.
7. Nahrazení hráčů počítačem simulovaným hráčem po jejich trvalém odpojení (ztrátě spojení), případně znovu napojení hráče, který byl už jednou nahrazen a znovu se mu podařilo připojit.
8. Textovou konverzaci hráčů.
9. Naplánování hry na určitý čas.

Práce bude obsahovat:
1. Popis použitých technologií.
2. Implementaci výše popsaného programu.
3. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30.6.2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 30.6.2017 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, zejména panu Ing. Davidu Ježkovi, Ph.D., za poskytnutá doporučení a rady a za čas strávený na společných konzultacích. Dále bych chtěl poděkovat své rodině za podporu, při tvorbě této bakalářské práce.

Abstrakt

Úkolem této bakalářské práce vytvořit Java herní portál, který bude mít část psanou na platformě JavaSE, tak i na platformě JavaEE. Seznámíme se s návrhem architektury práce, použitých technologií, návrhových vzorů a jejich využití. To vše doprovázené diagramy a kusy kódu pro lepší pochopení.

Klíčová slova: JavaSE, JavaEE, TCP Socket, EJB, JPA, JSF, relační databáze, vlákna, návrhové vzory

Abstract

The goal of this bachelor thesis is to create a Java game portal, which will be part of the JavaSE platform and the JavaEE platform. We will get acquainted with the design of architecture, used technologies, design patterns and their use. All the accompanying diagrams and pieces of code for better understanding.

Key Words: JavaSE, JavaEE, TCP Socket, EJB, JPA, JSF, relational database, threads, design patterns

Obsah

Seznam použitých zkratek a symbolů

JavaEE	–	Java Platform, Enterprise Edition
JavaSE	–	Java Platform, Standard Edition
JSF	–	JavaServer Faces
JSP	–	JavaServer Pages
EL	–	Expression Language
EJB	–	Enterprise Java Beans
JPA	–	Java Persistence API
DAO	–	Data access object

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je tvorba internetového herního portálu, který bude umožňovat hraní již vytvořených her, díky kterému máme možnost se blíže seznámit s technologiemi programovacího jazyka Java, volbu správných postupů a návrhových vzorů pro efektivní programování. Tuto práci můžeme rozdělit na dvě hlavní kapitoly. První část práce bude zaměřena na tvorbu desktopového a serverového rozhraní pomoci platformy JavaSE. Druhá část bude obsahovat webové rozhraní postaveno na platformě JavaEE.

V první části se budeme zaobírat komunikací mezi klientem a serverem pomoci socketů. Toto spojení budeme využívat pro veškerou komunikaci mezi samotnými herními klienty, herním serverem, serverem portálu, databází. Dále budeme řešit připojení a práci s databází pro manipulaci s uloženými daty (uživatelské údaje, záznamy o zápasech, turnajích a další).

Ve druhé části se budeme zabývat dynamickým webem, který bude sloužit pro zobrazování jednotlivých zápasů, turnajů a správu uživatelů.

Práce odhaluje komplexnější stránky programovacího jazyka Java. Teoretickým rozborem je snáze získáváno více praktických zkušeností. Tuto práci jsem si zvolil proto, protože jsem chtěl získat více zkušeností a nahlédnout více do hloubky jazyku Java. V této práci se blíže seznámíme s návrhem samotné architektury celé aplikace, použití knihoven pro práci s databází, s tvorba uživatelského rozhraní a tvorbou samotného jádra aplikací. Dalším faktorem pro výběr této práce bylo, že jsem již měl příležitost pracovat jako junior programátor v jazyce C#, a chtěl jsem zkusit něco nového a rozšířit si svůj obzor.

2 Technologie

Pro tuto práci je použita technologie Java Standart Edition neboli JavaSE, která slouží hlavně pro tvorbu desktopových aplikací v jazyce Java. Pro tvorbu grafického uživatelského rozhraní v desktopových aplikací je možnost výběru z těchto čtyř nejznámějších knihoven, a to Swing [1, 8], Standard Widget Toolkit (dále jen SWT) [2] a JavaFX [3, 8]. Dále byla použita technologie Java Enterprise Edition neboli JavaEE, která je primárně určená pro tvorbu webových aplikací a vychází právě z platformy JavaSE. Tyto aplikace ve většině případů běží na aplikačním serveru. Jako alternativu pro JavaEE lze použít také framewok Spring MVC, který stejně jako JavaEE staví na technologii Java Servlet [8].

2.1 Prezentační vrstva platformy JavaEE

Prezentační vrstvu u JavaEE je dále rozdělena na klientskou vrstvu a webovou vrstvu.

2.1.1 Klientská vrstva

Nejčastěji se jedná o webovou aplikaci, která je schopna zobrazit přijatá data obsahující tagy (HTML, XHTML ...). Klientská část webové aplikace se často nazývá jako tenký klient.

2.1.2 Webová vrstva

Pro realizaci webové prezentační vrstvy lze využít standartní technologie JavaEE JavaServer Pages (dále jen JSP) [5, 8] nebo novější JavaServer Face (dále jen JSF) [4, 8]. V případě použití Spring MVC lze použít framework Velocity, Freemarker, nebo Thymeleaf. Technologie prezentační vrstvy jak JavaEE, tak i Spring MVC však vychází z Java servletů.

- **Java servlety** - slouží pro zpracování příchozích dotazů. Nejčastěji se však setkáme s HTTP dotazy, kde je využit potomek servletu HTTPServlet, u kterého se volá jedna z metod doGet, doPost, které zpracovávají zmiňovaný HTTP dotaz, a dále zasílají odpověď (viz výpis 1).

```
public class MujServlet extends HttpServlet {
  @Override
  public void doPost(HttpServletRequest request, HttpServletResponse
      response) throws IOException {
      PrintWriter pw = response.getWriter();
      pw.println("<HTML><BODY>Hello world!</BODY></HTML>");
          pw.close();
  }
}
```

Výpis 1: Příklad Servletu

Frameworky technologie JavaEE.

- **JSP** - tato technologie slouží pro tvorbu statické a dynamické části webové stránky. Kde společně s JSP a JSTL tagy se používají Expression Language (dále jen EL) výrazy, pomoci kterých je možné zobrazit objekty ze session nebo z requestu, které připravila aplikační vrstva. Příklad použití JSP s EL výrazy, viz výpis 2.

```
<HTML>
  <BODY>
    <jsp:useBean id="datum" class="java.util.Date"/>
    Datum: ${datum.date}. ${datum.month}.${datum.year+1900}
  </BODY>
</HTML>
```

Výpis 2: Příklad JSP

- **JSF** - JSF je objektově založený framework, který slouží pro zobrazení a interakci s uživatelem za pomoci komponentů (tlačítek, textových polí ...), se kterými je možno pracovat na straně serveru. JSF je nástupcem právě JSP, který toto neumožňoval. Dále, také obsahuje objekty typu ManagedBean [6, 8] a stejně jako u JSP se používají pro přístup k datům EL výrazy. Příklad objektu typu ManagedBean viz výpis 3. Objektů typu ManagedBean může být více druhů, a to:
  - `@RequestScoped` existuje po dobu HTTP žádosti. Je vytvořena při žádosti a zase odstraněna při odpovědi.
  - `@ViewScoped` existuje tak dlouho, jak klient používá JSF náhled. Je odstraněn, pokud klient přejde na jiný JSF náhled.
  - `@SessionScoped` existuje po dobu jedné session.
  - `@ApplicationScoped` existuje tak dlouho, jak existuje webová aplikace. Vytvoří se při první HTTP žádosti a odstraní při vypnutí webové aplikace.

```
@ManagedBean
@RequestScoped
public class HelloWorld {
    @ManagedProperty(value = "#{message}")
    private Message message;
    ...
   }
```

Výpis 3: Příklad objektu typu ManagedBean

2.2 Prezentační vrstva platformy JavaSE

Pro tvorbu grafického uživatelského rozhraní je možnost využití těchto čtyř nejznámějších knihoven, a to Swing, Standard Widget Toolkit (dále jen SWT) a JavaFX.

- Swing (balíček `javax.swing.*`), jedná se o nástupce již téměř nepoužívaného grafického prostředí AWT vyvíjeno společností Sun Microsystems. Tato knihovna vznikla právě z AWT a podstatně ji vylepšila a rozšířila o nové komponenty. Je obsažena v Javě již od verze 1.2.
- JavaFX (balíček `javafx.*`), opět se jedná o produkt vyvíjený společností Sun Microsystems, který nahrazuje zmiňovaný Swing. Nově podporuje 3D grafiku, práci s animacemi, podporu multitouch, dotykových zařízení a měnitelnost vzhledu za pomoci CSS. Tato knihovna je součástí Javy od verze 7.
- SWT (balíček `org.eclipse.swt.*`), knihovna byla vytvořena společností IBM pro Eclipse IDE. K vykreslování využívá nativní knihovny na rozdíl od Swing nebo JavaFX. Klade důraz na rychlost a výkonost. Bohužel tato knihovna není obsažena již v Javě a je ji potřeba dodatečně přidávat. Hlavní nevýhodou této technologie je, že je platformě závislá (pro Windows, Linux, MacOS musí mít odlišné knihovny).

2.3 Aplikační vrstva

2.3.1 Aplikační vrstva JavaEE

Pro platformu JavaEE je možno využít technologii EJB [9, 10, 7, 8], která je jako JSF nativně součástí JavaEE. Technologie EJB je možné použít v případě aplikaci postavené na Spring MVC.

- EJB je standartní komponentní architektura, které se nachází na straně serverové. EJB komponenty mají za úkol oddělit právě datovou vrstvu od vrstvy prezentační. Prostředí, kde běží tyto komponenty, se nazývá EJB kontejner, který slouží pro izolování komponent od aplikace, a plní tyto funkce:
  - Řízení životního cyklu těchto komponent
  - Vzdálený přístup
  - Řídí transakce
  - Zaručuje souběžný přístup
  - Stará se o bezpečnost
  - Poskytuje přístup ke zdrojům

Existují tři typy EJB:

- *@Stateless* - Stateless session beans (také bezstavové beany) neuchovávají stavy mezi jednotlivými požadavky. Při každém požadavku je tvořena vždy nová instance.
- *@Stateful* - Stateful session beans (stavové beany) uchovávají svůj stav mezi požadavky klienta v rámci jedné session. Pro každého klienta se vytvoří na serveru instance stavového beanu.
- *@Singleton* - Singleton session beans (jedináček) slouží ke sdílení dat mezi klienty a komponentami. Využívá se zejména pro statické hodnoty.

Pro konstruktor beany je třeba použít anotaci u metody `@PostConstruct` z toho důvodu, že při vytváření třídy a zavolání konstruktoru beana není ještě inicializovaná.

2.3.2 Aplikační vrstva JavaSE

Aplikační vrstvou na platformě JavaSE je síťová komunikace mezi samotnými klienty a hrami, tak i s databází k práci s daty. Síťová komunikace mezi klienty, hrami nebo i s automatizovanými hráči je řešena pomocí soketů, který používá TCP síťový protokol. Díky tomuto druhu spojení je možnost jednoduchého přijímání a posílání serializovaných objektů, které mohou obsahovat požadavky nebo samotná data.

2.4 Datová vrstva

Datová vrstva pro platformy JavaSE i JavaEE zajišťuje framework Java Persistence API (dále jen JPA) [11, 8], který nám umožňuje práci s daty v databázi, nebo máme možnost použít přímé dotazování pomoci JDBC[12, 13].

- JPA - slouží k objektové relačnímu mapování (dále jen ORM), což je překlad objektů na tabulkovou reprezentaci a vytvoření vztahů a vazeb mezi nimi. Usnadňuje také provádění základních CRUD operací a minimalizuje používání SQL dotazů. JPA je dostupné jak pro platformu JavaEE, tak i pro JavaSE. V současné době nejznámější implementace jsou například Hibernate nebo EclipseLink. O entity se stará objekt třídy EntityManager, který má čtyři základní metody pro práci s objekty:
  - persist(entita) - uloží objekt do databáze
  - remove(entity) - odstraní objekt z databáze
  - merge(entity) - změní objekt v databázi
  - find(class, id) - vrací objekt, který souhlasí s class a má primární klíč id

  U JPA se často setkává s dědičností, která se řeší pomoci anotace `@Inheritance` nebo se může objevit tzn supertřída (označení třídy anotací `@MappedSuperclass`). V případě

použití supertřídy, tak děděná třída přebírá trvalé stavy a informace o mapování od supertřídy (nejčastěji se používá jako základní třída, která obsahuje primární klíč). Dědičnost za použití anotace `@Inheritance` může být třech typů:

  - `Single Table` - v tomto případě se vlastnosti třídy potomka a rodiče vloží do jediné tabulky, která bude sloužit pro všechny potomky.
  - `Joined` - pomocí tohoto typu dědičnosti se vytvoří tabulky jak potomka, tak i rodiče a jsou spojeny za pomoci primárního klíče.
  - `Table per class` - při použití tohoto typu se vytvoří tabulka pro každého potomka unikátní se všemi vlastnosti potomka a rodiče.

- JDBC - jedná se o rozhraní, které umožňuje přístup k datům uložené v relační databázi, ke kterým je přistupováno pomocí dotazů psané v SQL. V dnešní době se však již používá většinou s kombinací s ORM nástrojem.

3 Návrh aplikace

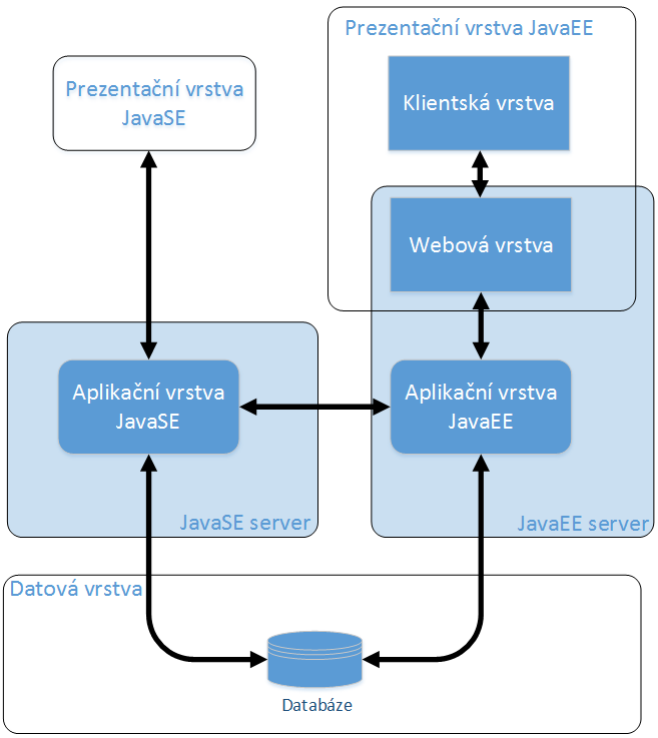
Práce je napsána v jazyce Java, konkrétně desktopová aplikace a server pomoci JavaSE. Webová část práce je vytvořena pomoci JavaEE, také za pomoci jednoduchého HTML a CSS.

Základem celé práce je vícevrstvá architektura, která se skládá z vrstvy prezentační, aplikační a datové (viz obrázek č. 1). Prezentační vrstva slouží jako viditelná část pro uživatele, která je zaopatřena vstupními daty a prezentací výsledků. V této práci tuto vrstvu představuje webová stránka, klientská aplikace nebo samotná hra. Další vrstva se nazývá aplikační nebo také funkční. Jedná se o prostřední vrstvu, která komunikuje jak s vrstvou prezentační, tak i s datovou. Tento oddíl přijímá data, nad kterými následně provádí operace a výpočty. Výsledné informace dále předává zpět. Tato část je považována jako zásadní, protože obstarává veškerou logiku aplikace. Zajišťuje ji server herního portálu a u webové části je realizována pomoci frameworku EJB. Poslední vrstva se nazývá vrstva datová, která obstarává persistenci dat. Obsahuje relační databázi a komunikaci s ní zajišťuje technologie Java Persistence API.

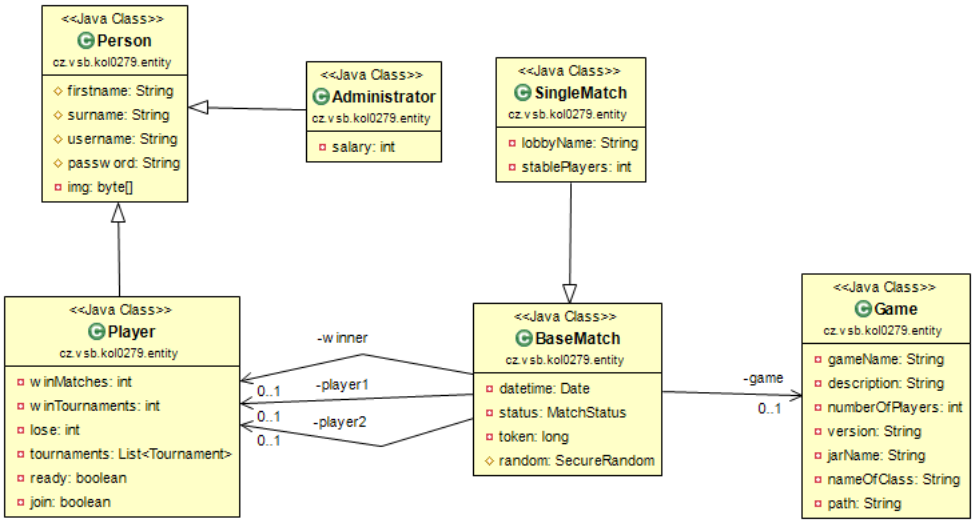
Mezi hlavní výhodu této architektury je, že každá vrstva je oddělena od druhé a nejsou na sobě závislé. To znamená, že jednotlivé vrstvy mohou být lehce nahrazeny nebo upraveny bez toho, aby ovlivnily chod celé aplikace. Také vývoj aplikace co využívá tuto architekturu může být znatelně rychlejší, protože jednotlivé vrstvy lze vyvíjet paralelně. Značnou nevýhodou však je hardwarova náročnost. Každá z těchto vrstev vyžaduje svůj hardware, což může být finančně náročné, ale také i náročné na údržbu.

Při tvorbě našeho herního portálu bylo třeba prvně vytvořit datovou vrstvu, která byla společná jak pro aplikaci na platformě JavaEE, tak i na JavaSE. Jako relační řídicí databázový systém se použil open source Apache Derby/Java DB, který má data uložená v relačních tabulkách. Byl navržen třídní diagram našeho herního portálu podle modelu 2 a 3. Kde první model obsahuje správu uživatelů, dostupné hry a rychlé zápasy. Druhý model přidává do herního portálu možnost turnajů.

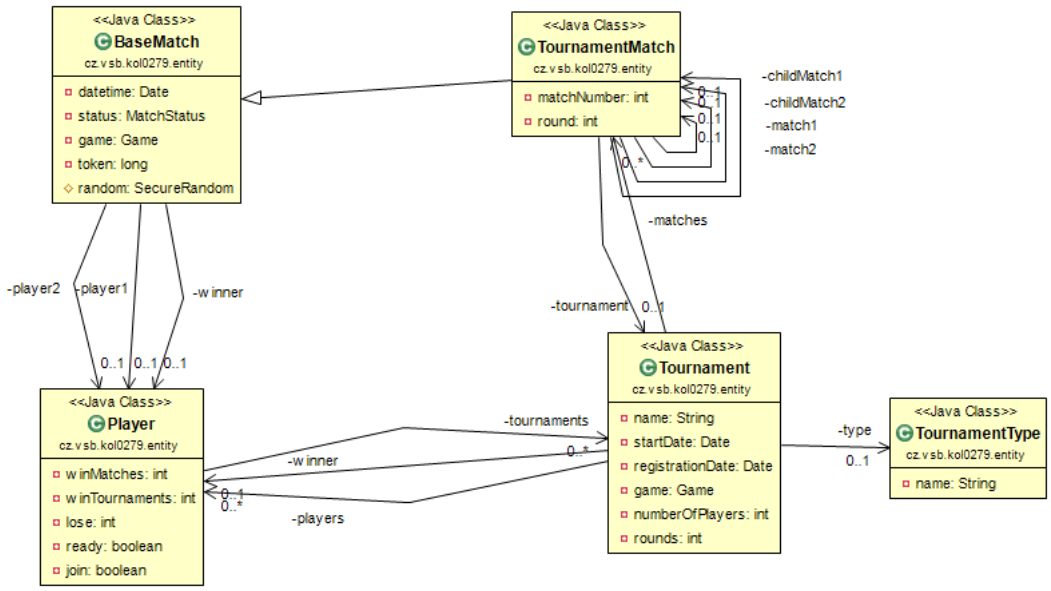
Jako poskytovatele JPA byl použit nástroj Hibernate k vytvoření struktury databáze. Strukturu je tvořena JPA třídami. Jedná se o obyčejnou třídu, u které je použita anotace `@Entity`. Příklad JPA třídy, viz výpis č. 4.



Obrázek 1: Architektura aplikace



Obrázek 2: Základní třídní diagram



Obrázek 3: Turnajový třídní diagram

```
@NamedQueries({ @NamedQuery(name = "Person.findAll", query = "Select e from
    Person e"),
      @NamedQuery(name = "Person.findByUsername", query = "Select e from Person
          e WHERE e.username = :username") })
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Person extends BaseEntity implements Serializable {
   protected String firstname;
   protected String surname;
   @Column(unique = true, updatable = true, insertable = true, nullable = true,
       length = 255)
   @Basic(optional = true)
   protected String username;
   @Column(unique = false, updatable = true, insertable = true, nullable =
      false, length = 32)
   @Basic(optional = true)
   protected String password;
   ...
```

Výpis 4: Příklad třídy Person v JPA

V JPA se může vyskytovat vedle vazeb 1:1, tak i vazby M:1, jako příklad se může vzít výpis č. 5, kde entita BaseMatch může mít jednoho vítěze, který je objekt typu Player. Naopak však

objekt typu Player může být vítězem ve více BaseMatch (zápasech). Mapování na cizí klíče zajišťuje parametr `targetEntity`, která určuje vlastníka této vazby.

```
public class BaseMatch extends BaseEntity implements Serializable {
   ...
   @ManyToOne(optional = true, targetEntity = Player.class)
   private Player winner;
   ...
}
```

Výpis 5: JPA tvorba vazby M:1

U JPA tříd se použije také dědičnost a to za pomoci `MappedSuperclass`, která se objevuje u základní třídy `BaseEntity`, viz výpis 6 a dědičnost typu `Joined`, která se používá pro dědění u tříd `Player` (příklad třídy, viz výpis 7) a `Administrator` z třídy `Person` (příklad třídy, viz výpis 4), výsledné databázové tabulky mohou vypadat dle obrázku 4.

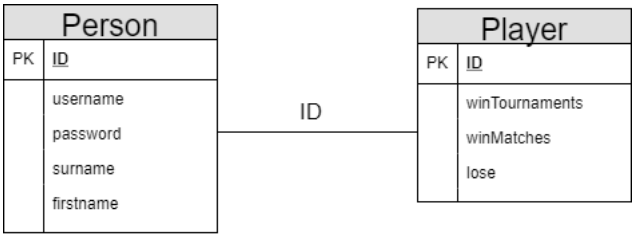
```
@MappedSuperclass
public class BaseEntity implements Serializable {
   ...
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long id;
   ...
   }
```

Výpis 6: Bázová třída BaseEntity

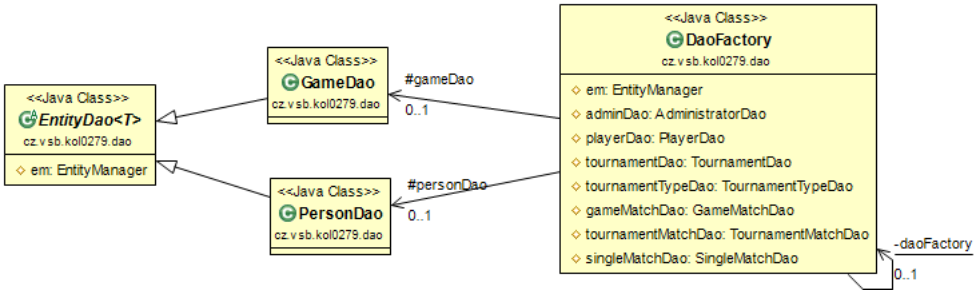
```
public class Player extends Person {
   ...
   @Column(unique = false, updatable = true, insertable = true, nullable =
      false)
   @Basic(optional = true)
   private int winMatches = 0;
   ...
   }
```

Výpis 7: Potomek třídy Person

K těmto vytvořeným JPA tříd, dle třídních diagramu, lze přistupovat pomocí návrhového vzoru Data Acces Object (dále jen DAO), který poskytuje specifické metody s daty a izoluje logiku správy dat. Společně s DAO je použit i návrhový vzor Singleton neboli jedináček, jenž zajistí právě jednu instanci návrhového vzoru DAO, a tak umožní globální přístup k databázi, viz diagram 5.



Obrázek 4: Databázové tabulky Person a Player



Obrázek 5: Návrhový vzor DAO a Singleton

Třída `DaoFactory` umožňuje přístup k jednotlivým DAO třídám, které obsahují potřebné metody pro práci s daty dané entity v relační databázi, také dědí z abstraktní generické třídy, která jim poskytuje základní CRUD operace. V této fázi se práce rozděluje na tvorbu rozhraní na platformě JavaEE a rozhraní na platformě JavaSE.

4 Tvorba webového serveru

4.1 Aplikační server

Rozhraní na platformě JavaEE je sdružení knihoven, které potřebují běhové prostředí. To zajišťují takzvané aplikační servery. Ty mohou být od mnoha výrobců. Nejznámější open source jsou například:

- GlassFish
- WildFly

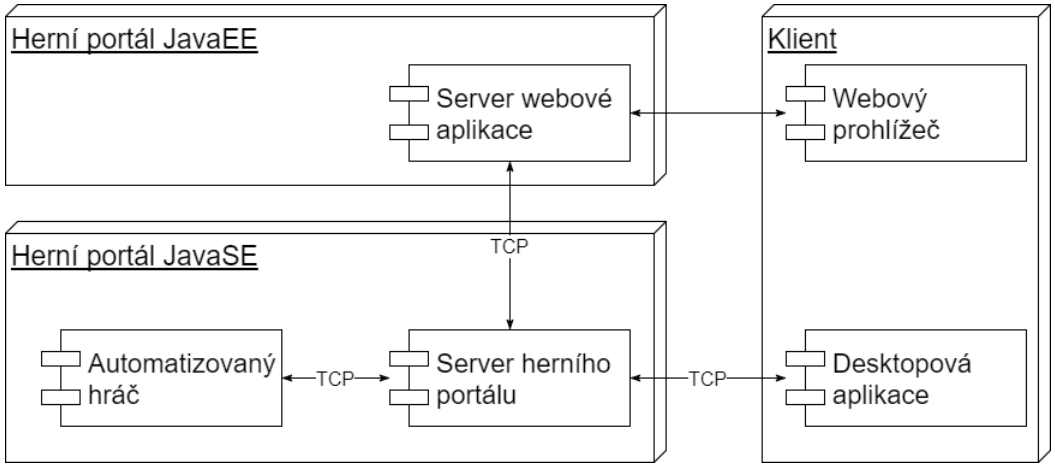
V práci je použit právě aplikační server WildFly, který je třeba nejdříve nakonfigurovat tak, aby správně komunikoval s relační databází.

4.2 Aplikační vrstva a webová vrstva

Na WildFly serveru běží aplikační vrstva, která je tvořena EJB komponenty. V konstruktoru těchto komponent se získá instance DaoFactory pro přístup k databázi. Je potřeba také EntityManager pro práci s entitamy. Ten je dosažen pomoci anotace `@PersistenceContext`. Příklad takové EJB komponenty viz výpis 8.

```
@Stateless
public class PersonSB implements PersonSBLocal {
   @PersistenceContext
   protected EntityManager em;
   protected DaoFactory dao;
   @PostConstruct
   public void PersonSB() {
      dao = DaoFactory.getInstance(em);
      init();
   }
   @Override
   public void init() {
      dao.getPersonDao().init();
   }
   @Override
   public Person login(String email, String passwd) {
      return dao.getPersonDao().login(email, passwd);
   }
   ...
```

Výpis 8: Příklad Session beany PersonSB



Obrázek 6: TCP komunikace mezi JavaEE aplikací a JavaSE

Ve webové aplikaci se využívá také připojení k serveru herního portálu pomoci TCP socketu, aby se získaly informace, které nejsou uloženy v databázi, ale jen na serveru (viz diagram 6). Může se jednat například o aktuálním skóre právě probíhající hry (viz výpis 9), seznamu probíhajících her, turnajů a další. Připojení k serveru z platformy JavaEE je totožné jako z platformy JavaSE.

```
public String getActualScore(long token) {
   try {
      ClientCommunicator server = new ClientCommunicator(
            new Socket(ServerConnection.getHostname(), ServerConnection.
               getPort()));
      server.sendObject(new GetActualScore(token));
      Object obj = server.readObject();
      if (obj instanceof GetActualScore) {
         GetActualScore data = (GetActualScore) obj;
         server.disconnect();
         if (data.getScore() != null)
            return data.getScore();
      }
      return null;
   } catch (IOException e) {
      e.printStackTrace();
      return null;
   }
}
```

Výpis 9: Metoda pro získání aktuálního skóre

K získání těchto dat je zapotřebí nejdříve navázat spojení se serverem vytvořením nové instance třídy Socket, který se předá instanci třídy `ClientCommunicator`. Ta zajišťuje přijímání a odesílání dat na server. Na tento server se zašle instance třídy dotazu (například GetActualScore, kde tato třída požaduje token zápasu). Server přijatý objekt přetypuje na rozhraní určené pro dotazy z webu (`WebPerformer`) a zavoláme na tento objekt metodu perform, která vyžaduje jako parametr objekt, který implementuje metody z rozhraní `WebCommand` určené pro webové dotazy, a provede se požadovaná metoda v našem případě GetActualScore, viz výpis 10.

```
@Override
public void GetActualScore(long token) {
   for (ServerMatch match : client.getServer().getServerMatches()) {
      if (match.getToken() == token) {
         GetActualScore receiveCmd = new GetActualScore(match.getActualScore
             ());
         client.sendObject(receiveCmd);
         return;
      }
   }
   client.sendObject(null);
}
```

Výpis 10: Metoda GetActualScore na straně serveru

Server prochází všechny aktuálně hrané zápasy a hledá ten, který souhlasí s přijatým tokenem. Když jej nalezne server herního portálu, dotáže se instance herního serveru probíhajícího zápasu. Po získání dat server zasílá data zpět webové aplikaci.

Základem aplikační vrstvy na platformě jsou EJB komponenty, které jsou schopny pracovat s daty mezi webovým aplikačním serverem a relační databází. Aby uživatelům tato data byla zobrazena a mohli s nimi pracovat, je potřeba vytvořit webovou vrstvu, kde je využita technologie JSF, která obsahuje objekty typu ManagedBean (příklad viz výpis 11), pomocí kterých jsou zobrazena data uživatelům.

```
   @ManagedBean
   @SessionScoped
public class LoginMB {
   @EJB
   protected PersonSBLocal SBLocal;
   protected Person loggedPerson = null;
   protected Person editedPerson = null;
   protected Administrator editedAdmin = null;
   protected Player editedPlayer = null;
   protected String username;
   protected String passwd;
   public String login() {
      loggedPerson = SBLocal.login(username, passwd);
      if (loggedPerson != null)
         return "index";
      FacesMessage message = new FacesMessage("Chyba pri prihlasovani", "Spatne
          uzivatelske jmeno nebo heslo");
      FacesContext.getCurrentInstance().addMessage(null, message);
      return "";
   }
```

Výpis 11: Příklad ManagedBeany pro přihlášení

Pro vytvoření objektu typu ManagedBean je zapotřebí přidat k třídě anotaci `@SessionScoped`, která zařídí, že objekt typu ManagedBean existuje po dobu jedné session. V tomto případě, aby zůstal uživatel přihlášený. Dále anotace `@EJB` dovoluje přístup k EJB komponentám, v tomto příkladě k objektu typu SessionBean, která implementuje rozhraní PersonSBLocal. Metody s návratovými hodnotami typu String slouží pro přesměrování na jinou stránku, v případě prázdného řetězce nedochází k přesměrování.

Pro tvorbu stránek pro uživatele se používá XHTML. K přístupu k objektům typu ManagedBean jsou použity výrazy EL. Příklad přihlašovací stránky viz výpis 12.

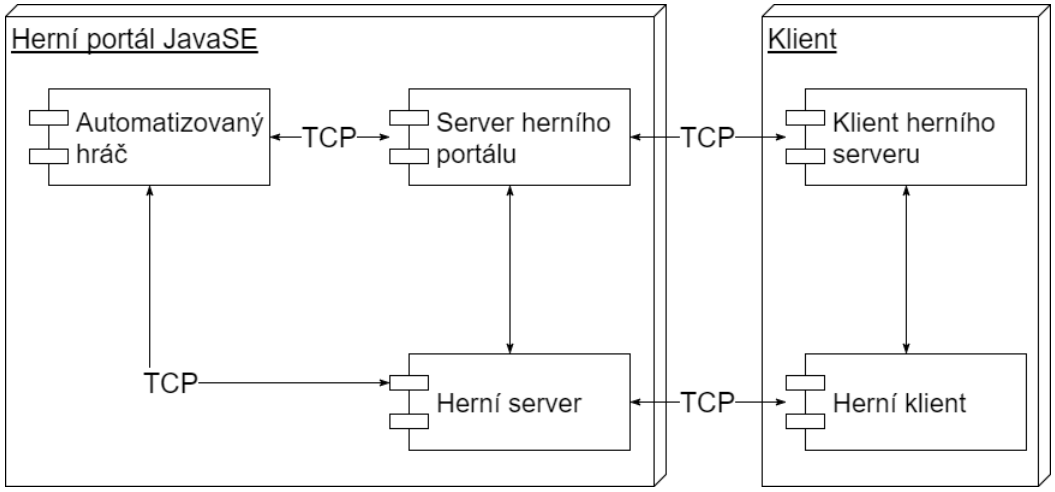
```
   <h:form rendered="#{!loginMB.logged}">
   <h:messages />
   <h:panelGrid columns="2">
      <h:outputText value="Uzivatelske jmeno" />
      <h:inputText value="#{loginMB.username}" required="true" />
      <h:outputText value="Heslo" />
      <h:inputSecret value="#{loginMB.passwd}" required="true" />
   </h:panelGrid>
   <h:commandButton value="Prihlasit se" action="#{loginMB.login()}" />
   <h:commandButton value="Zaregistrovat se"
      action="#{loginMB.registration()}" />
</h:form>
```

Výpis 12: XHTML přihlašovací stránka s použití EL

K objektům typu ManagedBean lze přistupovat pomocí názvu třídy typu Bean s malým písmenem, pokud však objekt typu ManagedBean je pojmenovaný (@ManagedBean(name="PrihlaseniMB")), jsou odkazovaní podle pojmenovaného jména.

5 Tvorba serveru herního portálu

Server herního portálu obstarává komunikaci mezi klienty a zpřístupňuje komunikaci mezi herním serverem a herním klientem. Komunikace mezi nimi je řešena pomocí TCP socketů, viz diagram 7.



Obrázek 7: TCP komunikace serveru herního portálu

Server přijímá připojení a o každého připojeného klienta se stará vlastní vlákno, kde je možno nalézt i metody pro čtení příchozích dat a odesílání, tato klientská třída se nazývá `ServerCommunicator`.

Server neustále naslouchá a čeká na připojení klienta. Při navázání spojení je získáván socket klienta, který následně je předán nové instanci třídy `ServerCommunicator`. O každý nový objekt se stará vlákno tak, aby nezatěžovalo hlavní vlákno, kde běží server, viz výpis 13.

```
private List<ServerCommunicator> clients = new ArrayList<>();

   public static void main(String[] args) throws Exception {
      new Server(new ServerSocket(2222)).getConnections();
   }
   public void getConnections() throws IOException {
      while (true) {
         Socket socket = listen.accept();
         ServerCommunicator comm = new ServerCommunicator(socket, this);
         comm.start();
         addCommunicator(comm);
      }
   }
```

Výpis 13: Životní cyklus serveru

Při vytvoření instance této třídy a obdržení socketu klienta od serveru se v konstruktoru vytvoří streamy:

- `ObjektInputStream`, který slouží pro příjem objektů od serveru.
- `ObjektOutputStream` k odesílání objektu serveru.

Následně je spuštěno samotné vlákno, které neustále kontroluje ObjektInputStream, zda neobdržel data, viz výpis 14.

```
while (handle) {
   Object obj = null;
   obj = input.readObject();
   if (obj != null) {
      if (obj instanceof ServerPerformer)
         ((ServerPerformer) obj).perform(clientCommands);
      if (obj instanceof WebPerformer)
         ((WebPerformer) obj).perform(webCommands);
   }
}
```

Výpis 14: Čtení příchozích dat od klienta

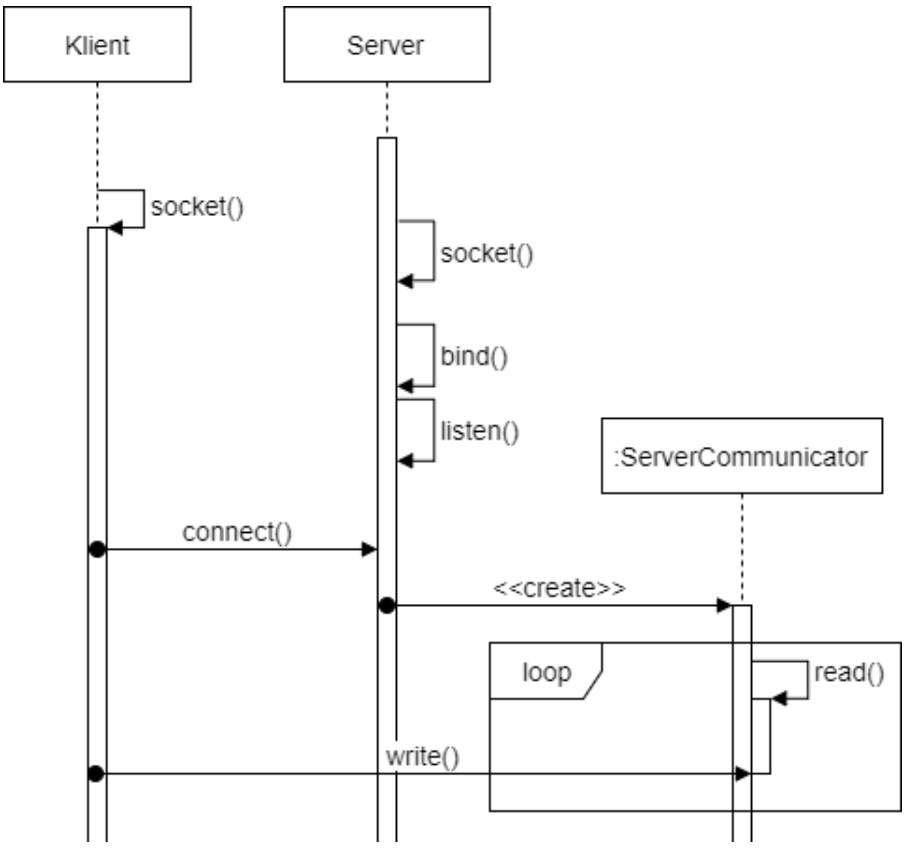
Životní cyklus a připojení klienta k serveru a metoda pro čtení dat je zobrazena na obrázku 8.

Příchozí objekt je třída implementující rozhraní s metodou perform, která má jeden parametr, a to rozhraní obsahující seznam potřebných metod. Tento příchozí objekt je přetypován na potřebné rozhraní a je zavolána metoda perform, které je předána třída, která implementuje právě rozhraní s metodami, které se mají provést. Objekt typu:

- ServerPerformer slouží pro provedení metod, příchozí ze strany klientské
- WebPerformer slouží pro provedení metod ze strany webové

5.1 Generování turnajových zápasů

Generování zápasových dvojic nastává až po uplynutí data startu turnaje, tedy až turnaj bude mít stálý počet hráčů a nebude možno se k turnaji již přihlásit. Příkaz pro vytvoření rozpisu turnaje nastává až tehdy, když některý uživatel zobrazí detail turnaje. Tento příkaz se provádí jen v případě, že se jedná o první navštívení detailu turnaje. V současné době náš herní portál disponuje jen dvěma typy turnaje.



Obrázek 8: Sekvenční diagram čtení a zasílání dat na serveru

- `Každý s každým` - Algoritmus tohoto turnaje viz výpis 15. Pro tvorbu rozpisu tohoto turnaje je zapotřebí sudý počet hráčů, proto v případě lichého počtu, je přidán nový hráč, který bude představovat volnou hru. Ve finálním rozpisu však zápasy, kde se tento přidaný hráč nachází neobjevují. K tomu, aby se vytvořil tento turnaj, je potřeba si vytvořit kopii seznamu hráčů a odstranit hráče na první pozici(proměnná players obsahuje všechny hráče a players2 obsahuje hráče bez prvního). Základem celého algoritmu jsou dva cykly, kde jeden je vnořen do druhého.

```
for (int i = 0; i < players.size() - 1; i++) {
      if (players2.get(i).getId() != -1 && players.get(0).getId() != -1)
          {
         p1 = players2.get(i);
         p2 = players.get(0);
      }

      for (int idx = 1; idx < halfSize; idx++) {
         int firstTeam = (i + idx) % players2.size();
         int secondTeam = (i + players2.size() - idx) % players2.size();
         if (players2.get(firstTeam).getId() != -1 && players2.get(
             secondTeam).getId() != -1) {
            p1 = players2.get(firstTeam);
            p2 = players2.get(secondTeam);
         }
      }
   }
```

Výpis 15: Turnaj každý s každým

- `Playoff pavouk` - Při tvorbě playoff pavouka může nastat jedna situace, že počet hráčů v turnaji není číslo mocniny 2. To způsobuje, že někteří hráči postupují automaticky do druhého kola. Je zapotřebí si vypočítat, kolik dvojic hráčů budou hrát v prvním kole.

$$x = N - P/2$$

Pomocí této rovnice se vypočítá, kolik dvojic hráčů budou hrát v prvním kole, zbylí hráči postupují do druhého kola automaticky. Kde N je počet hráčů v turnaji a P je nejblíže výše postavená mocnina 2 k počtu hráčům. Tím se vytvoří dvě skupinky hráčů. Jedna, jak byla již zmíněna, bude obsahovat dvojice hráčů a druhá hráči bez oponenta. Druhé kolo se vytvoří tak, že promíchají tyto dvě skupiny, aby se utkali i hráči, kteří automaticky postoupili s těmi, co hráli v prvním kole, a vytvoří se dvojice jako v kole prvním. Nyní

již vychází, že počet dvojic je mocnina čísla 2. A lze pomoci cyklu projít až do finálové dvojky.

5.2 Spuštění hry

Před spuštěním hry je nutno, aby hráči byli oba připravení ke hře. To schválí pomoci tlačítka v lobby. Lobby dále obsahuje barevnou indikaci o aktuálním stavu hráče, viz obrázek 12:

- Šedá barva hráče - znázorňuje, že hráč se nenachází v lobby
- Bíla barva hráče - hráč je v lobby, ale není připraven ke hraní
- Zelená barva hráče - hráč je v lobby a také je připraven ke hraní a čeká na oponenta

Jakmile jsou hráči připraveni (průběh kontroly připojených hráčů viz UML diagram 9), předá se token klientovi, podle kterého je identifikována instance serveru hry. Tuto instanci reprezentuje objekt třídy `ServerMatch`. Ten obstarává samotný životní cyklus herního serveru, předává mu hráče, nahrazuje automatizovaným hráčem, reaguje na opuštěné hráče a získává data o aktuálním dění hry. Po přijetí tokenu na klientské straně je naváženo nové spojení se serverem, které se předá samotné hře, aby mohla samostatně komunikovat se serverem. Tímto spojením je zaslán serveru token a informace o uživateli. Ten podle tokenu nalezne instanci třídy ServerMatch a předá se ji informace o připojení ke hře. Jakmile se do objektu ServerMatch přidají všichni hráči, dynamicky se načte herní server a vytvoří se jeho instance, které jsou předány informace o hráčích. Podobný proces je proveden na straně klientské, kde dynamicky je načten herní klient a předá se mu informace pro komunikaci s herním serverem. Pro komunikaci se samostatnou hrou a herním klientem je použito rozhraní, které musí implementovat. U herního klienta to je `IGame` rozhraní, viz výpis 16, které obsahuje jedinou metodu. A to pro spuštění hry, která obsahuje parametry username (uživatelské jméno) a objekt typu ClientCommunicator (obsahuje input streamy, output streamy a socket pro komunikaci se serverem).

```
public interface IGame {
   void startGame(String username, ClientCommunicator server);
}
```

Výpis 16: Rozhraní herního klienta

A pro herní server se jedná o `IGameServer`, viz výpis 17

```
public interface IGameServer {
   void addClient(String username, Socket client, ObjectOutputStream output,
       ObjectInputStream input);
   void reconnectClient(String username, Socket client, ObjectOutputStream
       output, ObjectInputStream input);
   void addAI(String disconnectedClient, Socket server, Socket client,
       ObjectOutputStream outputServer,
         ObjectOutputStream outputClient, ObjectInputStream inputServer,
             ObjectInputStream inputClient);
   Queue<String> requestAI();
   String actualScore();
   void startServer();
   void stopServer();
   String getWinner();
}
```

Výpis 17: Rozhraní herního serveru

Interface herního serveru obsahuje metodu pro přidání uživatele, znovupřipojení, přidání automatizovaného hráče, získání aktuálně odpojených uživatelů, aktuálního skóre, spuštění a ukončení serveru a získání vítěze.

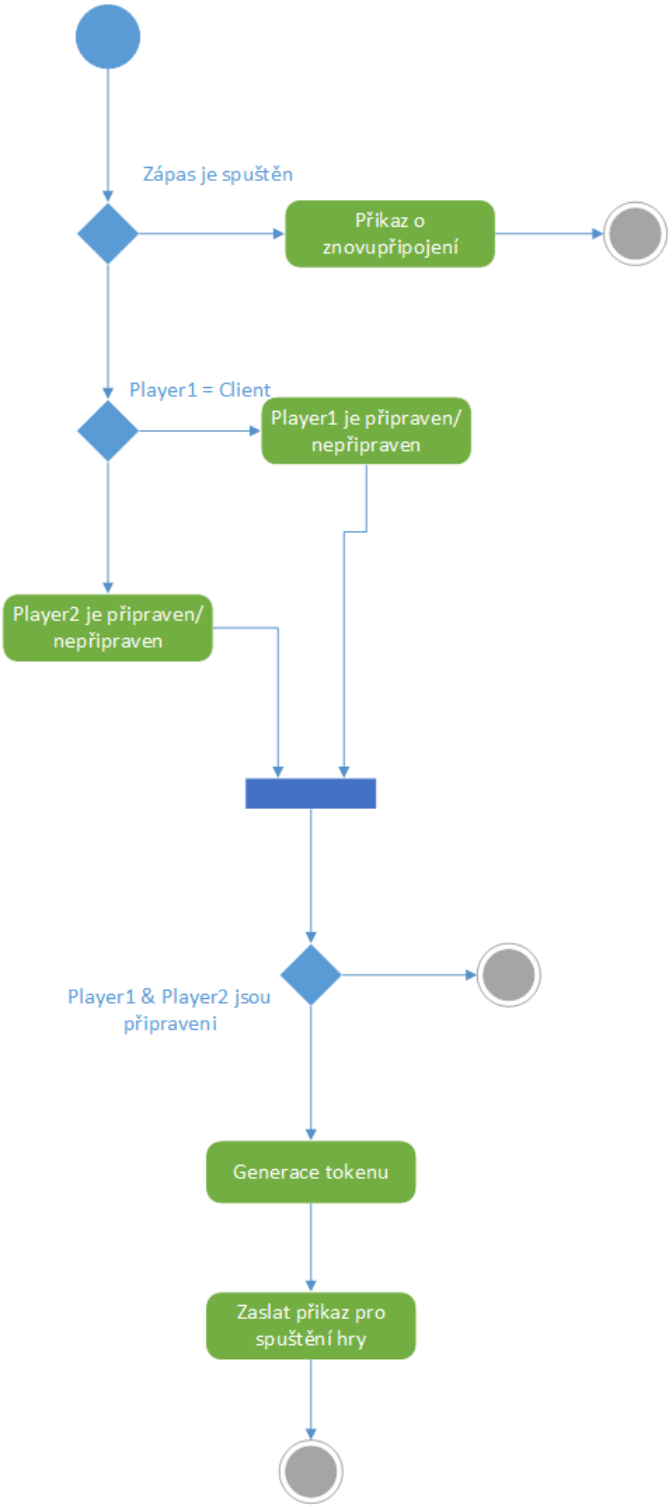
5.3 Odpojení hráče a znovupřipojení

Odpojené uživatele zajišťuje fronta, která se nachází na serveru hry. Do této fronty při odpojení uživatele se vloží uživatelské jméno odpojeného. A ze strany serveru herního portálu pomoci metody `Queue<String> requestAI()` se získá fronta odpojených uživatelských jmen. Tato uživatelská jména jsou postupně přebírána a následně vytvoříme nové TCP spojení se serverem a identifikuje se toto spojení zasláním tokenu hry a uživatelského jména opuštěného. Nyní jsou k dispozici nástroje pro komunikaci s automatizovaným hráčem jak ze strany serveru, tak i ze strany klienta, který se však nachází také na serveru. Tyto informace jsou posílány zpět hernímu serveru, který si již sám implementuje hraní automatizovaného hráče. Kód nahrazení hráče simulovaným hráčem, viz výpis 18.

```
        Iterator iterator = serverMatch.getGameServer().requestAI().iterator();
        while (iterator.hasNext()) {
           String disconnected = serverMatch.getGameServer().requestAI().remove
               ();
           System.out.println("ODPOJEN " + disconnected);
           ClientCommunicator aiClient;
           try {
              aiClient = new ClientCommunicator(new Socket("localhost", 2222));
              aiClient.sendObject(new IdentifySocketCommand(serverMatch.
                  getToken(), disconnected, true));
              aiClient.sendObject(new StopReadSocketCommand());
              for (Map.Entry<String, ServerCommunicator> ail : serverMatch.
                  getpList().entrySet()) {
                 if (ail.getKey().equals(disconnected)) {
                    serverMatch.getGameServer().addAI(disconnected, ail.getValue
                        ().getSocket(),
                          aiClient.getSocket(), ail.getValue().getOutput(),
                              aiClient.getOutput(),
                          ail.getValue().getInput(), aiClient.getInput());
                    break;
                 }
              }
           } catch (IOException e) {
              e.printStackTrace();
           }
        }
```

Výpis 18: Nahrazení simulovaným hráčem

Znovupřipojení hráče probíhá totožně jako spuštění hry a vytváření nového spojení. Jen s tím rozdílem, že hra, ke které se uživatel pokouší připojit, je ve stavu běžící (MatchStatus.Running), a tak přímo objekt třídy `ServerMatch`, který obdrží informace o nově vytvořeném spojení, které byly provedeny na klientské části, která předá rovnou existující instanci herního serveru pomoci metody reconnectClient(String username, Socket client, ObjectOutputStream output, ObjectInputStream input). UML diagram znovupřipojení, viz obrázek 9.



Obrázek 9: Kontrola připravených hráčů

6 Tvorba klientské aplikace

6.1 Tvorba rychlé hry a turnajů

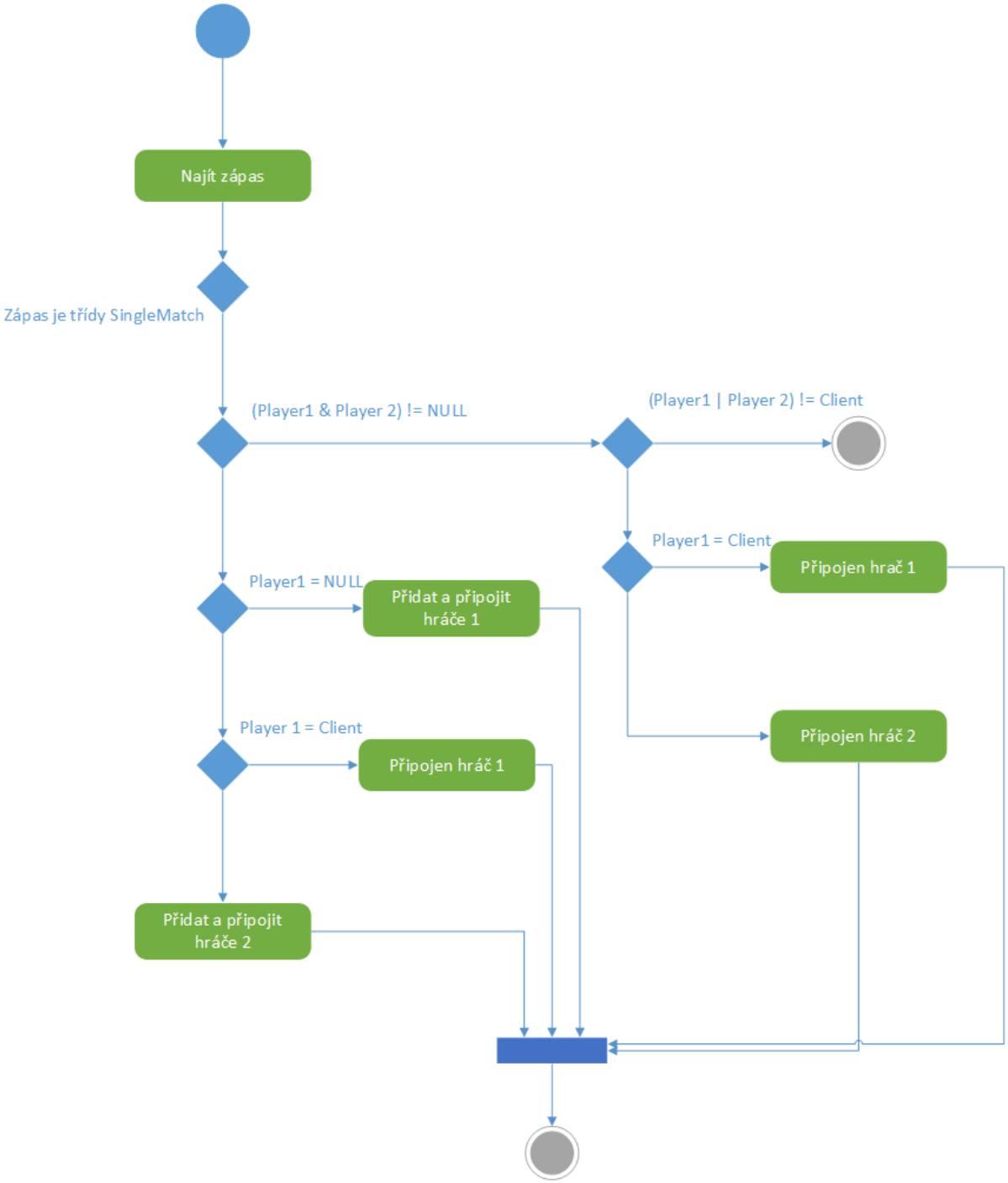
Pro tvorbu rychlé hry je potřeba hru, která musí být přidána administrátorem v herním portálu. Náhled grafického rozhraní pro tvorbu rychlé hry, viz obrázek 11. Tvorba rychlé hry obsahuje pole oponent, pokud pole zůstane nevyplněné, hra bude otevřená pro hráče a může se kdokoliv připojit. Odložený start rozhoduje, zda bude hra okamžitě založena, zůstává na serveru a neukládá se do databáze, dokud se hra nedohraje, při volbě opožděného startu se hra uloží do databáze a hráči se nabídne po uplynutí zvoleného času. Při okamžitém založení se generuje záporný token a nastavuje se jako ID. Tvorba turnajů je velice podobná jako u rychlých her, jen se turnaj vždy ukládá do databáze z důvodu generování zápasů.

6.2 Připojení ke hře

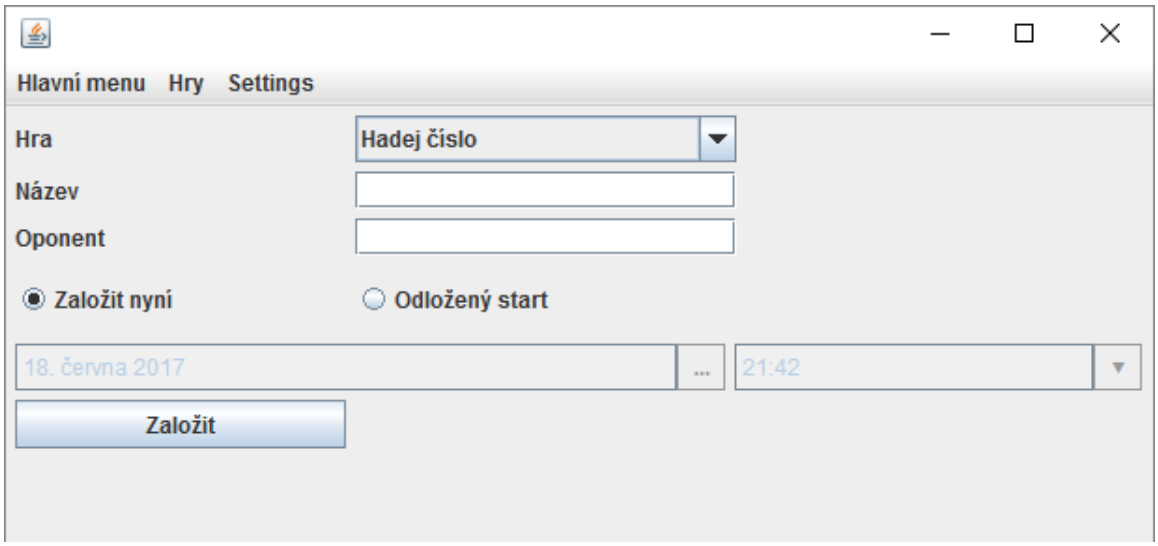
Objekty hráčů obsahují vlastnosti `ready` a `join`, které jsou typu boolean. Tyto vlastnosti vypovídají o hráči, zda je připojen do lobby (join = true) a zda je hráč v lobby připraven hrát (ready = true). Při připojování do hry probíhají kontroly uživatele, zda ve hře je místo, které navíc není určeno pro jiného hráče (při tvorbě byl zvolen jiný oponent), viz UML diagram 10. Po úspěšném připojení do lobby se hráči změní vlastnost join na true. Připojování u turnajových zápasů je téměř totožné. Jen s tím rozdílem, že pokud oponent neexistuje na pozici Player1 nebo Player2 (Player1 == null nebo Player2 == null), pak je nutno hráče dohledat v předešlém turnaji na pozici vítěze, pokud však i vítěz neexistuje (winner == null), pak se k zápasu nelze připojit, protože nebyl odehrán předchozí zápas a je potřeba počkat na vítěze.

6.3 Textová konverzace

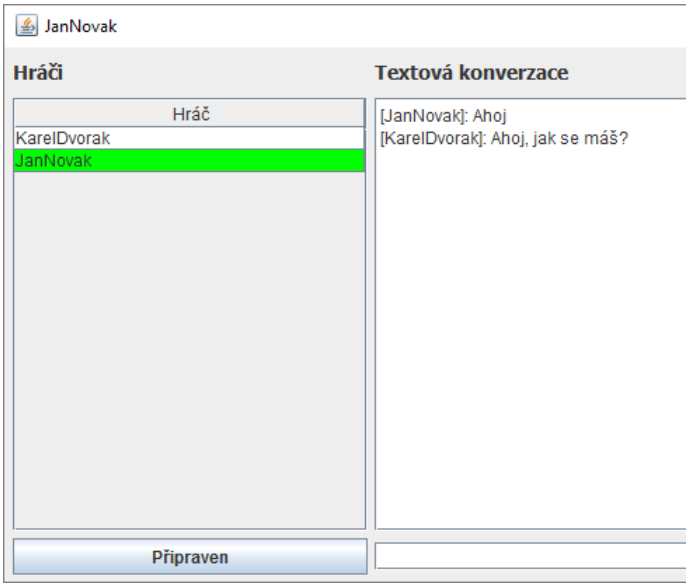
Textová konverzace hráčů je řešena v lobby hry, viz obrázek 12. To umožňuje komunikaci jen mezi hráči, kteří se nachází v totožném lobby. Konverzaci je možno využívat jak před spuštění samotné hry, tak i v průběhu hraní.



Obrázek 10: UML připojení do rychlé hry



Obrázek 11: Tvorba rychlé hry



Obrázek 12: Textová konverzace a indikace stavu hráčů

7 Závěr

V této bakalářské práci sem znázornil jednu z cest, jak docílit vytvoření svého herního portálu jak za pomoci platformy JavaEE, díky které jsme byli schopni vytvořit webovou aplikaci pro prezentaci výsledků a správu uživatelů, tak i za pomoci platformy JavaSE, pomoci které jsme vytvořili jádro samotného herního portálu, které zprostředkovávalo rozhraní pro hraní her a komunikaci mezi uživateli. Velice mi také pomohly konzultace se svým vedoucím práce, který mě dokázal nasměrovat správným směrem a docílit, tak výsledné práce. Díky této práci jsme získal ohromné množství zkušeností a nových znalostí, které bych dále mohl využít jak v profesním životě, tak ve svých vlastních projektech.

V rozšiřování s tímto projektem bych chtěl v budoucnu rád pokračovat. Z důvodu poznávání nových technik, návrhových vzorů, technologií a zdokonalování v jazyce Java, který je velice obsáhlý, jak jsem se již ujistil při psaní této bakalářské práce.

Literatura

[1] Swing. In: Wikipedia FI MUNI [online]. Brno: Fakulta informatiky Masarykovy univerzity, 2001- [cit. 2017-06-24]. Dostupné z: https://kore.fi.muni.cz/wiki/index.php?title=Swing

[2] WILSON, Mike. SWT: the standard widget toolkit. Boston: Addison-Wesley, 2004-. ISBN 978-0321256638.

[3] Úvod do JavaFX. Itnetwork.cz [online]. Čápka, c2017 [cit. 2017-06-24]. Dostupné z: https://www.itnetwork.cz/java/javafx/java-tutorial-uvod-do-javafx

[4] Java Server Pages. In: Wikipedia FI MUNI [online]. Brno: Fakulta informatiky Masarykovy univerzity, 2001- [cit. 2017-06-24]. Dostupné z: https://kore.fi.muni.cz/wiki/index.php?title=Java_Server_Pages.

[5] BURD, Barry. JSP: Javaserver pages, podrobný průvodce. Praha: Computer Press, 2003. ISBN 978-807-2268-047.

[6] The Java EE 6 Tutorial. Oracle Documentation [online]. California: Oracle, 2013 [cit. 2017-06-24]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/girch.html

[7] Technologie Enterprise Java Beans. TIS - Tvorba informačních systémů [online]. Ostrava: Beneš [cit. 2017-06-24]. Dostupné z: http://www.cs.vsb.cz/benes/vyuka/tis/prednasky/09-ejb.pdf

[8] JAT - Java Technologie. JAT - Java Technologie [online]. Ostrava: Ježek, c2015 [cit. 2017-06-24]. Dostupné z: http://swi.cs.vsb.cz/dam/jcr:ce5ed61d-5553-4a06-bfe8-c98681372aad/JAT_prednasky.pdf

[9] Enterprise Beans. Oracle Documentation [online]. Oracle, c2010 [cit. 2017-06-24]. Dostupné z: https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cp38/index.html

[10] The Business Tier of the Java EE Architecture. Uc3m [online]. Madrid: Pickin, Mendoza, Madrid, c2011 [cit. 2017-06-24]. Dostupné z: http://ocw.uc3m.es/ingenieria-telematica/communication-software/slides/the-business-tier-of-the-java-ee-5-architecture

[11] Persistence. Oracle Documentation [online]. Oracle, c2013 [cit. 2017-06-24]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html

[12] Úvod do JDBC. Interval.cz [online]. Šeda, 2003 [cit. 2017-06-24]. Dostupné z: https://www.interval.cz/clanky/uvod-do-jdbc/

[13] Lesson: JDBC Introduction. Oracle Documentation [online]. Oracle, c1995-2015 [cit. 2017-06-10]. Dostupné z: https://docs.oracle.com/javase/tutorial/jdbc/overview/